成绩：_____

评语：

华南理工大学广州学院

2015-2016 学年度第 2 学期

课程名称： C 语言大作业

题　　目： 超大整数运算算法的 C 语言描述与实现

专业班级： 15 级软件工程一班

年　　级： 2015

姓　　名： 区绍彬

学　　号： 201510098097

班内序号： 20

摘要　高精度大整数计算算法是通过将超大数据拆分成多个小项，利用基础的加减乘运算进行计算，避免了因为基本数据类型所分配的空间无法储存所有的数据而造成无法计算或者精度丢失的问题。

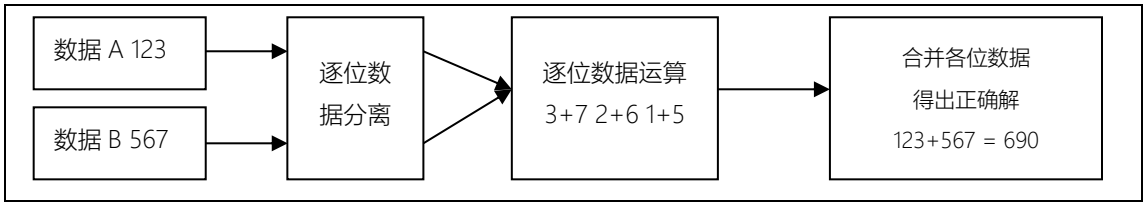
一、序言

由于计算机高级程序语言在设计的时候，考虑到内存空间分配的合理性，大部分计算机程序设计语言的基本数据类型的有效范围一般不超过-2^{64}~2^{64}。但随着数据的规模的增长，大数据技术的发展；以及在非对称加密、科学研究、机器学习等领域的超大型数据的广泛应用，使得设计计算机程序处理数据时的难度会加大。所以，我们根据分治算法的思想，设计了以下的超大整数运算算法（加、减、乘法）。

二、算法的具体实现

考虑到算法程序的可移植性，我们选用的是可以在 Linux 和 Windows 平台下通用的 C 语言编译器 GCC。而考虑到本算法在实际研究和生产应用环境中使用，数据的进制我们采取了十进制。如果使用的数据序列为二进制，八进制或者十六进制，我们可以先使用进制转换算法进行转换后再进行运算，抑或修改程序中进位相关的数据实现。

我们从中小学生笔算常常会用到的竖式计算中得到灵感：学生在进行竖式运算的时候，会把复杂的运算转换成两个小于十的数的运算，逐位求解，最后对求出的数据进行合并，得出正确的解答。而我们可以用分治算法的基本思想来描述这个过程，对于规模大的数据，我们将他分解为多个小规模的问题，通过求解小规模的问题，得到最终的正确解。因此我们把小学生计算的过程抽象成算法。



图一 大数运算算法原理

对于超大的整数 A($A > 2^{64}$)，我们约定 A 的数字位数为 n，即 A 可以表示为:

$$A = \sum_{i=0}^{n} a_i \times 10^i$$

其中 a_i 所表示的是 A 中每一个具体的数位，同理，对于超大型整数 B，我们可以将它表示为：

$$B = \sum_{i=0}^{n} b_i \times 10^i$$

因此，我们可以用以下的过程对超大型的整数进行运算操作：

算法 分治法处理超大型整数

Step1：序列 A 进行分离，提取出 A 中的每一个$a_i(i \geq 0)$。同样的，对数据序列 B 也采用相同的操作，提取出对应的$b_i(i \geq 0)$。

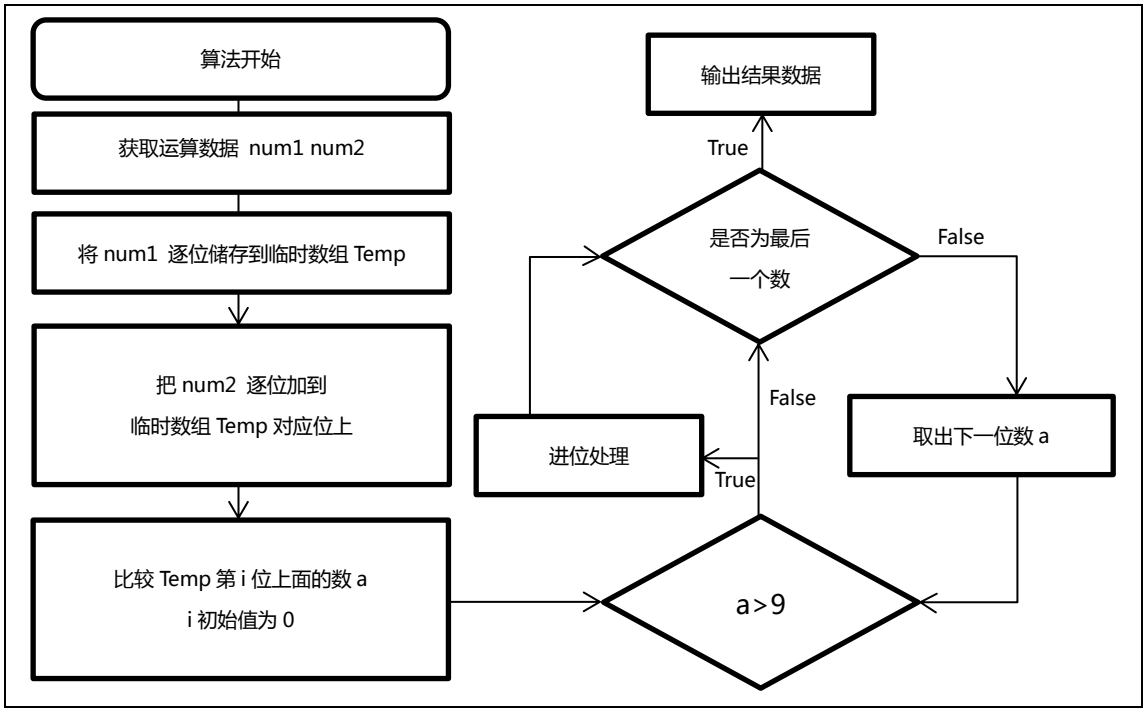
Step2：根据需要对各个位进行操作，例如处理加法时，会把序列 A 和序列 B 对应的位数相加，并存入数据序列（$c_i = a_i + b_i$）。

Step3：根据加减乘运算的特性，把序列 C 合并成答案序列并返回。

由于在实际运算中，加法，减法，乘法所对应的数位变化处理会有所不同，我们会在下文进一步展开讨论。

2.1 加法

如图二所示，处理超大数组的加法时，我们采用的是逐位相加，全部计算完毕之后再进行进位操作的策略。



图二 加法算法流程

根据流程图，我们可以编写出对应的大数加法算法，关键代码[1]如下(程序清单 1):

```
int length_1=strlen(num_1),length_2=strlen(num_2);
int i,maxlen;
maxlen=length_1>length_2?length_1:length_2; //获取最大长度
maxlen++;
memset(temp,0,sizeof(temp));  //把全数组初始化
for(i=1;i<=length_1;i++)
        temp[i]=num_1[length_1-i]-'0';
for(i=1;i<=length_2;i++)
        temp[i]+=num_2[length_2-i]-'0';
for(i=0;i<=maxlen;i++)
    if(temp[i]>9){
        temp[i]-=10;
        temp[i+1]++;
    }
for(i=maxlen;i>1;i--)
    if(temp[i]==0) maxlen--;
      else break;
 for(i=0;i<maxlen;i++)
    res[i]=temp[maxlen-i]+'0';
 res[i]='\0';
```

程序清单 1 大数加法

在处理结束时，因为临时变量数组中的数据顺序都是反向的，因此我们需要将临时变量

[1] 程序清单中 num_1 num_2 res 数组均为字符串数组，下同

的数据顺序反转，变成最终的结果。

2.2 减法

减法的处理和加法还是有一点的区别的，比如在借位运算以及正负数的处理上。同样的，我们采用和加法一样的模拟小学生进行竖式运算来进行求解。

获取到数据 num1，和数据 num2 之后，我们会对两个数进行对比。

如果 num1≥num2,那么 num1-num2 的结果严格大于 0，此时把 num1 和 num2 逐位相减并做借位就可以了；假如出现 num1<num2 的情况，那么结果严格小于 0，此时我们可以把 num1-num2 看作是-(num1-num2)=num2-num1。

进行大数比对的函数如程序清单二，先进行位数的比对，位数较多的数为较大数，若两个数的长度一致，那么就从被减数的最高位开始向下比对。

因为假定存在位数相同的正整数 a,b ，$a = \sum_{i=0}^{n_a} a_i \times 10^i$, $b = \sum_{i=0}^{n_b} b_i \times 10^i$。现令 c=a-b，又因 $n_a=n_b$，令 $n=n_a=n_b$，易求得$c = \sum_{i=0}^{n} (a_i - b_i) \times 10^i$。

此时我们不妨假设$i = i_n(n \geq 0)$且$(a_i - b_i) > 0$。此时，我们很容易证得 c>0。同理，我们也可以证得当 b 的最高位大于 a 的最高位时，c<0。

算法 超大整数大小比较算法

Step1: 比较两者的长度 若 num1 的长度大于 num2 的长度 则判断为 num1>num2，反之，则 num2>num1。此时可直接返回结果给调用函数。

Step2: 如果 num1 的长度和 num2 一致，则进行最高位的判断，如果 num1 的最高位大于 num2 的最高位，那么判定 num1 较大，如果 num1 的最高位和 num2 的最高位一致，那么依次判定次高位，第三高位等，直至判断出结果或者数据遍历结束。

根据以上阐述，我们可以把比较过程用 C 语言描述出来（程序清单二）：

```
Int numCompare(char num_1[],char num_2[]){
    int length_1 = strlen(num_1);
    int length_2 = strlen(num_2);
```

```
    int i,flag;
    if(length_1>length_2) return 0;
        else if(length_2>length_1) return 1;
            else{
                for(i=0;i<=length_2;i++){
                    if(num_2[i]<num_1[i]){
                        flag=0;   break;
                    }else if(num_2[i]>num_1[i]){
                        flag =1;   break;
                    }
                }
                if(flag) return 1;
                  else return 0;
            }
}
```

程序清单二 大数比对算法

判断完成后，减法的运算和加法的相似。以 num1>num2 为例：

算法 大数的加法

Step1：把 num1 逐个数位存储进临时数组 temp 中，然后把 temp 中各个数位减去 num2 中对应的数位。

Step2：运算结束后，进行统一的借位运算操作（程序清单三）。

在竖式运算中，我们减法如果出现被减数大于减数的情况，我们会向更高的一位借 10，以使结果每一位数字都是 10 以内的正整数。

我们的算法也不例外，我们会对 temp 中的数字进行逐个判断，如果出现负数，则会给这个数加上 10，并且让更高一位减去 1。这样就可以实现借位的运算。

同样的，若 num2>num1，我们只需要把运算符两侧对调，num2 变成减数，num1 变成被减数，最后处理数据时加上负号即可。

因此，对应过程的关键代码如程序清单三：

```
void dec(char num_1[], char num_2[], char res[]){
    int temp[MAXN];    //我们约定 MAXN 为 1000
    int length_1=strlen(num_1),length_2=strlen(num_2);
    int i,check,maxlen;
    maxlen=length_1>length_2?length_1:length_2;
```

```
    maxlen++;
    memset(temp,0,sizeof(temp));
     check = numCompare(num_1,num_2);
     if(check){
          for(i=1;i<=length_2;i++)
               temp[i]=num_2[length_2-i]-'0';
          for(i=1;i<=length_1;i++)
               temp[i]-=num_1[length_1-i]-'0';
          for(i=0;i<=maxlen;i++)
               if(temp[i]<0){
                    temp[i]+=10;
                    temp[i+1]--;
               }
          for(i=maxlen;i>1;i--)
               if(temp[i]==0)maxlen--;
                    else break;
          res[0]='-';
          for(i=0;i<maxlen;i++)
               res[i+1]=temp[maxlen-i]+'0';
          res[i+1]='\0';
     }else{
          for(i=1;i<=length_1;i++)
               temp[i]=num_1[length_1-i]-'0';
          for(i=1;i<=length_2;i++)
               temp[i]-=num_2[length_2-i]-'0';
          for(i=0;i<=maxlen;i++)
               if(temp[i]<0){
                    temp[i]+=10;
                    temp[i+1]--;
               }
          for(i=maxlen;i>1;i--)
               if(temp[i]==0)maxlen--;
                    else break;
          for(i=0;i<maxlen;i++)
               res[i]=temp[maxlen-i]+'0';
          res[i]='\0';
     }
}
```

程序清单三 大数减法

2.3 乘法

处理乘法的时候，我们可以把原理描述成有整数 a 和整数 b。其中 n 位整数 a 满足 $a=\sum_{i=0}^{n} a_i \times b \times 10^i$。因此我们在进行运算的时候需要对每组运算结果乘上 10.由于我们在

设计的时候是按照数位进行数据储存，因此在大数乘法运算时，我们提出以下的思路：

算法 大数乘法

Step1：整数 b 取出最低位，然后分别和数 a 中的各个数位相乘，并且把各个数位的运算结果存储在临时数组 temp 中，其中第一位（最小位）存储在下标为 k（k 的初值为 0）的数组单元中。

Step2：取出 b 的次低位，和 1 中一样，分别相乘，并和 temp 中的数位相加，值得注意的是，因为需要乘以 10，所以 k 首先自增 1，然后将 step2 求出的第一位（最小位）数存储在下标为 k 的数组单元中。

Step3：循环执行 Step2，直至执行完成，进行逐个数位进位处理，并返回计算结果

根据上述算法描述，我们可以编写程序清单四所示的 C 语言程序进行求解

```
void muti(char num_1[],char num_2[],char res[]) {
    int i,j,k=0,c=0,s;
    int length_1 = strlen(num_1);
    int length_2 = strlen(num_2);
    int temp[MAXN];   //我们约定 MAXN 为 1000
   memset(temp,0,sizeof(temp));
    memset(res,0,sizeof(res));
    for(i=1;i<=length_2;i++){
        for(k=i,j=1;j<=length_1;j++){
            temp[k++] += (num_2[length_2-i]-'0')*(num_1[length_1-j]-'0');
        }
    }
    for(i=0;i<k;i++){
        temp[i]+=c;
        s=temp[i];
        temp[i] = s%10;
        c=s/10;
    }
    for(i=k-1;i>0;i--){
        res[k-1-i] = temp[i]+'0';
    }
}
```

程序清单四 大数乘法

大数乘法的关键在于进位运算，我们采取了类似于竖式运算中向左移动、相加的方式，

避免了复杂的数学运算，实现了节省空间和时间的高精度整数乘法。

三、结语

通过模拟竖式运算，可以设计出运算超过千位的超大型整数，这种运算描述简洁，便于维护，对使用者数学的基础要求较低。而且可以适应大部分的超大型整数运算。但这种算法具有一定的局限性，例如在空间复杂度和处理乘法时的时间复杂度较高。下一步的研究是应用高等数学和线性代数中的傅里叶变换、矩阵运算等理论对算法的性能和算法的运算效率进行有效的提升。

参考文献

[1] 刘汝佳.算法竞赛入门经典[M].北京. 清华大学出版社.2009:71